Hybrid Techniques Based on Solving Reduced Problem Instances for a Longest Common Subsequence Problem

Christian Blum[1] Maria J. Blesa[2]

[1] Artificial Intelligence Research Institute (IIIA-CSIC)
UAB Campus, Bellaterra, Spain
(christian.blum@iiia.csic.es)

[2] Computer Science Department
Technical University of Barcelona – BarcelonaTech
UPC North Campus, Barcelona, Spain
(mjblesa@cs.upc.edu)

Abstract

Finding the longest common subsequence of a given set of input strings is a relevant problem arising in various practical settings. One of these problems is the so-called longest arc-preserving common subsequence problem. This NP-hard combinatorial optimization problem was introduced for the comparison of arc-annotated Ribonucleic acid (RNA) sequences. In this work we present an integer linear programming (ILP) formulation of the problem. As even in the context of rather small problem instances the application of a general purpose ILP solver is not viable due to the size of the model, we study alternative ways based on model reduction in order to take profit from this ILP model. First, we present a heuristic way for reducing the model, with the subsequent application of an ILP solver. Second, we propose the application of an iterative hybrid algorithm that makes use of an ILP solver for generating high quality solutions at each iteration. Experimental results concerning artificial and real problem instances show that the proposed techniques outperform an available technique from the literature.

Keywords: combinatorial optimization, longest common subsequences, integer linear programming, heuristic, hybrid algorithm

1 Introduction

In computer science terms, a *string* (or sequence) x of length l_x is a finite sequence of characters from a finite alphabet Σ. In fact, strings are popular data types for representing and storing information. Words and even complete texts, for example, may be stored in a computer in terms of strings. However strings are not only useful in fields such as information and text processing. They arise, in particular, in the field of computational biology. The reason is that most of the genetic instructions involved in the growth, development, functioning and reproduction of living organisms are stored by means of *Deoxyribonucleic acid* (DNA) and *Ribonucleic acid* (RNA) molecules, which are either double-stranded (DNA) or single-stranded (RNA) sequences of nucleotides. In short, each nucleotide is composed of a nitrogenous base, a five-carbon sugar (ribose or deoxyribose), and at least one phosphate group. Concerning RNA, each nucleotide has one of four different nitrogenous bases: guanine (G), uracil (U), adenine (A), and cytosine (C). As a consequence, any RNA molecule can be represented as a string of symbols from $\Sigma = \{G, U, A, C\}$, which is called the *primary structure* of a RNA molecule. The primary structure of a RNA molecule is a simplified representation, because RNA molecules fold in space and different nucleotides bind together, for example, by means of hydrogene bonds. Generally, guanine (G) can only bind with cytosine (C) and uracil (U) can only bind with adenine (A). These hydrogene bonds are present in the so-called *secondary structure* of an RNA molecule; see Figure 1a for an example.

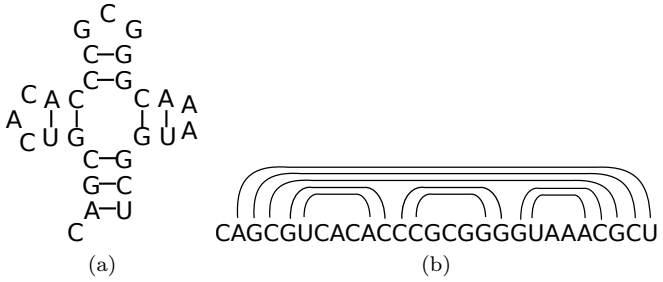


Figure 1: (a) Example of the secondary structure of an RNA molecule. (b) The corresponding arc-annotated sequence. The example is reproduced from [1].

For computer science purposes, the hydrogene bonds of the secondary structure of an RNA sequence x can be represented by a so-called *arc annotation set* P_x. In technical terms, P_x is an unordered set of pairs of positions of a string x.[1] Each pair $(i_1, i_2) \in P_x$ represents an arc between positions i_1 and i_2 and is called an *arc annotation*. The only convention is that $i_1 < i_2$ must hold for any arc $(i_1, i_2) \in P_x$. Finally, i_1 is called the *left endpoint* of arc (i_1, i_2), and i_2 is called the *right endpoint*. A pair (x, P_x) is called an *arc-annotated sequence* [2] (or arc-annotated string). Given this definition, note that the secondary struc-

[1] As a convention, the positions of a string x range from 1 to l_x.

ture of an RNA sequence can conveniently be described by an arc-annotated sequence; see Figure 1b for an example. In fact, arc-annotated sequences have been widely used for this purpose (see, for example, [3]). In particular, arc-annotated sequences have shown to be useful for the structural comparison of RNA sequences. One of the usual measures when comparing two (or more) sequences is the length of their *longest common subsequence* (LCS); see, for example, [4, 5]. In this context, given a sequence x over a finite alphabet Σ, sequence t is called a *subsequence* of x, if t can be produced from x by deleting characters. Given a set of input strings $\{s_1, \ldots, s_n\}$, the problem of finding the longest commons subsequence of all input strings is, in general, NP-hard [6]. The best techniques available nowadays for solving this problem are based on beam search [7] (see [8], for example).

1.1 The LAPCS Problem

The longest common subsequence problem in the context of arc-annotated sequences—the *longest arc-preserving common subsequence* (LAPCS) problem—has first been introduced in [9, 2]. Given two input sequences x and y, the set of possible *assignments* A is defined as the set of all $a_{i,j}$—where $i \in \{1, \ldots, l_x\}$ and $j \in \{1, \ldots, l_y\}$—such that $x[i] = y[j]$. In other words, A consists of all $a_{i,j}$ such that at position i of x and at position j of y there is the same letter. A valid common subsequence of the two input sequences x and y can then be represented by a subset $S \subseteq A$ that fulfills the following conditions:

- **Common subsequence condition:** For any two assignments $a_{i,j}, a_{k,l} \in S$ (where $a_{i,j} \neq a_{k,l}$) it must hold that either $i < k$ and $j < l$, or $i > k$ and $j > l$.

In order to translate such a solution into the corresponding common subsequence, the assignments in S have to be ordered from small to large indices, either according to the first or the second index. Then, the letters corresponding to the assignments must be joined in this order.

A solution S that fulfills the common subsequence condition is called *arc-preserving* if the arcs induced by the solution are preserved:

- **Arc preservation condition:** for any two assignments $a_{i,j}, a_{k,l} \in S$ (where $a_{i,j} \neq a_{k,l}$ and $i < k$) it must hold that $(i, k) \in P_x \Leftrightarrow (j, l) \in P_y$.

Given two arc-annotated input strings (x, P_x) and (y, P_y), the LAPCS problem consists in finding a solution $S \subseteq A$ that fulfills both the common subsequence and the arc preservation condition and is of maximal cardinality. Note that such a mapping corresponds to the longest arc-preserving common subsequence of x and y.

In practise, the nature of the arc annotation in the context of RNA sequences generally satisfies some conditions. Given an arc-annotated string (x, P_x), the

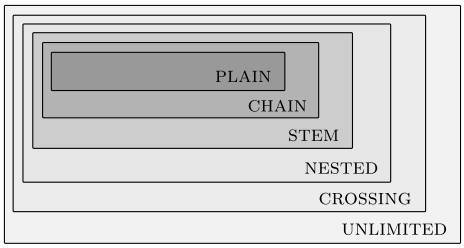


Figure 2: Hierarchy of different classifications of arc-annotated sequences.

relative positioning of two arcs (i_1, i_2) and (i_3, i_4) from P_x, who do not share any endpoint, is completely described by three binary relations.

1. The *precedence relation* ($<$): $(i_1, i_2) < (i_3, i_4)$ if $i_2 < i_3$
2. The *embedding relation* ($\sqsubset$): $(i_1, i_2) \sqsubset (i_3, i_4)$ if $i_1 < i_3$ and $i_4 < i_2$
3. The *crossing relation* ($\between$): $(i_1, i_2) \between (i_3, i_4)$ if $i_1 < i_3 < i_2 < i_4$

Based on these relations, six levels of arc structure have been considered in the literature: UNLIMITED (no restriction at all), CROSSING (there is no character incident to more than one arc), NESTED (there is no character incident to more than one arc and no arcs are crossing), STEM (there is no character incident to more than one arc, and given any two arcs, one is embedded into the other), CHAIN (there is no character incident to more than one arc, no arcs are crossing, and no arc is embedded into another one), and PLAIN (there is no arc). The resulting hierarchy of arc-annotated sequences is shown in Figure 2. Different versions of the LAPCS problem can then be denoted as follows: LAPCS$(\cdot, \cdot)$ where each of the two dots must be replaced by the classification of the arc annotation of the first and the second input string. For example, in problem LAPCS(UNLIMITED,NESTED), the arc annotation of the first input string is classified as UNLIMITED, and the one of the second one as NESTED. In this paper, however, we deal with the most general version of the problem, LAPCS(UNLIMITED,UNLIMITED). For simplicity reasons we refer to this problem version simply as LAPCS.

Finally, note that the LAPCS problem is a sub-problem of the more general EDIT problem [10], which is of considerable practical interest.

1.2 Existing Work

Different versions of the LAPCS problem have been studied so far in the literature, especially for what concerns classical complexity and parameterized complexity results. In [2, 9], for example, it was shown that the most general problem version—that is, LAPCS(UNLIMITED,UNLIMITED)—is NP-hard. Moreover, it is well known that LAPCS(PLAIN,PLAIN) can be solved in polynomial time with the dynamic programming algorithm by Smith and Waterman [11].

Table 1: NP-hard cases of the LAPCS problem. The first two table columns indicate the characterizations of the two input strings, without any order.

First characterization	**Second characterization**	Complexity
UNLIMITED	UNLIMITED	NP-hard [2, 9]
UNLIMITED	CROSSING	NP-hard [2, 9]
UNLIMITED	NESTED	NP-hard [2, 9]
UNLIMITED	CHAIN	NP-hard [2, 9]
UNLIMITED	PLAIN	NP-hard [13]
CROSSING	CROSSING	NP-hard [2, 9]
CROSSING	NESTED	NP-hard [2, 9]
CROSSING	CHAIN	NP-hard [2, 9]
CROSSING	PLAIN	NP-hard [13]
NESTED	NESTED	NP-hard [13]
STEM	STEM	NP-hard [14]

Table 2: Polynomially solvable cases of the LAPCS problem. The first two table columns indicate the characterizations of the two input strings, without any order.

First characterization	**Second characterization**	Complexity
NESTED	CHAIN	$\mathcal{O}(nm^3)$ [15, 1]
NESTED	PLAIN	$\mathcal{O}(nm^3)$ [15, 1]
CHAIN	CHAIN	$\mathcal{O}(nm^3)$ [15, 1]
CHAIN	PLAIN	$\mathcal{O}(nm)$ [2, 9]
PLAIN	PLAIN	$\mathcal{O}(nm)$ [11]

A nice overview concerning complexity and parameterized complexity results can be found in [12]. Moreover, the most important results are summarized in Table 1 and Table 2. Finally, it is important to note that the only heuristic from the literature that is applicable to the most general case of the LAPCS problem was proposed in [1]. A detailed description of this heuristic is provided in Section 3.1.

1.3 Contribution of this Work

Note that this work is a significant extension of a preliminar paper [16]. We first describe the LAPCS problem in form of an integer linear program (ILP) [17]. This ILP model follows the same idea as other existing ILP models for related LCS problems (see, for example, [18]). Due to the fact that this ILP model does not allow the application of a general purpose ILP solver, not even when small problem instances are concerned, different ways of using this model in a heuristic way are studied. First, the model is reduced—in a heuristic, deterministic

way—and then solved by CPLEX.[2] The size of the model reduction is shown to be a crucial parameter in this context. Furthermore, the ILP model is exploited in an iterative hybrid algorithm that makes use of an operator known from *optimal solution merging* in evolutionary algorithms. This is a concept which refers to merging at least two parent solutions and applying an exact technique for finding the best possible solution in this union. Such an approach was—in the context of evolutionary algorithms—first described in [19] for the independent set problem. In [20], this concept is applied to the quadratic assignment problem, and in [21] to the so-called k-cardinality tree problem. More recent applications can be found in [22] for a supply management problem, and in [23] for permutation problems in general. Theoretical studies of such operators were presented in [24, 25, 26]. Note that the proposed algorithm—due to the employed solution merging operator—may also be seen as a large neighborhood search technique [27]. Finally, the extension in comparison to [16] concerns that heuristic model reduction technique and a comprehensive experimental evaluation of the proposed techniques.

1.4 Outline of the Paper

The remainder of this paper is structured as follows. Section 2 presents a description of the existing heuristic from the literature. In Section 3 we outline an ILP model for the tackled problem and we present a heuristic based on this ILP model. This heuristic is based on reducing the size of the model before solving it by means of a general purpose ILP solver. Next, the proposed hybrid algorithm is outlined in Section 4. Finally, an extensive experimental evaluation both concerning artificial and real problem instances is provided in Section 5. Moreover, a discussion and an outlook to future work is given in Section 6.

2 Existing Heuristic for LAPCS

The only heuristic from the literature that is applicable to the most general version of the LAPCS problem was described in [1]. This heuristic works as follows. First, the dynamic programming algorithm by Smith and Waterman [11] is applied to input strings x and y, disregarding the arc annotations. This results in a mapping $S \subseteq A$ that possibly violates some of the arc preservation constraints. The following procedure is applied in order to *repair* this invalid solution. First a graph G is constructed whose vertex set consists of a vertex v for each assignment $a_{i,j} \in S$. Two vertices v (corresponding to an assignment $a_{i,j} \in S$) and v' (corresponding to an assignment $a_{k,l} \in S$ with $i < k$) are connected by an edge in G if either $(i, k) \in P_x$ or $(j, l) \in P_y$, but not both. In words, two vertices of G are connected by an edge if they represent a violation of the arc preservation

[2] IBM ILOG CPLEX is an optimization software package which includes state-of-the-art exact techniques for solving integer linear programming models, among others. It is available for free for academic purposes. For more information we refer the interested reader to `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`.

constraints. The goal is now to remove as few assignments from S as possible until S is a valid solution. For doing so, the *maximum independent set* (MIS) problem [28] is solved in G, with a subsequent removal of all assignments from S that correspond to vertices that are not in the optimal solution to the MIS problem. In our implementation we used CPLEX to solve the MIS problem in all cases. This deterministic heuristic is henceforth referred to as HEURISTIC.

3 An ILP Model for the LAPCS Problem

The LAPCS problem can be stated in terms of an ILP model in the following way. First, the model consists of a binary variable $z_{i,j}$ for each possible assignment $a_{i,j} \in A$. The set of all binary variables is henceforth denoted by Z. It is said that a pair of variables $z_{i,j}, z_{k,l} \in Z$ (with $z_{i,j} \neq z_{k,l}$ and $i \leq k$) are *in conflict* with each other, if setting both variables to one violates (1) the common subsequence condition, (2) the arc preservation condition, or both. In technical terms, two variables $z_{i,j}, z_{k,l} \in Z$ (with $i \leq k$) are in conflict, if at least one of the following holds:

1. $j \geq l$
2. Either $(i,k) \in P_x$ or $(j,l) \in P_y$, but not both at the same time.

The LAPCS problem can then be rephrased as the problem of selecting a maximal number of non-conflicting variables from Z. Given these notations, the ILP is stated as follows.

$$\textbf{max} \sum_{z_{i,j} \in Z} z_{i,j} \tag{1}$$

subj. to:

$$z_{i,j} + z_{k,l} \leq 1 \quad \forall\, z_{i,j} \neq z_{k,l}, i \leq k \text{ in conflict} \tag{2}$$

$$z_{i,j} \in \{0,1\} \quad \text{for } z_{i,j} \in Z \tag{3}$$

Hereby, constraints (2) ensure that at most one variable can be selected for the solution from each pair of variables that are in conflict.

3.1 Heuristic Reduction of the ILP Model

Due to a large number of variables and constraints, this model is probably not very useful in practice, at least not for the application of a general purpose ILP solver. In fact, as will be outlined later, CPLEX was not able to provide a valid solution for any of the problem instances tackled in this paper within a reasonable computation time limit. Therefore, we considered a heuristic way of reducing the model size before applying an ILP solver, based on the following intuition: it is rather unlikely that a variable $z_{i,j}$ will take value one in the optimal solution if, for example, i is very small and j is very large, or vice versa.

In words, if i refers to a position rather at the beginning (resp., the end) of input string x and j refers to a position rather at the end (resp., the beginning) of input string y, it is very unlikely that the corresponding assignment forms part of an optimal solution. With this idea in mind we can formalize the following way of reducing the size of the ILP model. Let l denote the length of the shorter one of the two input strings, that is, $l := \min\{l_x, l_y\}$. And let $0 \leq p \leq 100$ denote a fixed *separation factor*. Then, we define the *maximum gap* $\text{gap}_{\max}$ as follows:

$$\text{gap}_{\max} := \max\left\{1, \left\lfloor \frac{p \cdot l}{100} \right\rfloor\right\} \tag{4}$$

Given $\text{gap}_{\max}$ we can reduce the set of assignments (A) that corresponds to a problem instance in the following way:

$$A^r := \{a_{i,j} \in A \mid |i - j| \leq \text{gap}_{\max}\} \tag{5}$$

In other words, all assignments $a_{i,j} \in A^r$ must fulfill the condition that the gap between i and j is at most $\text{gap}_{\max}$. The reduced ILP which uses A^r instead of the complete set A of assignments is henceforth denoted by $\text{ILP}^r(p)$. Moreover, the application of an ILP solver to a reduced model can be seen as an ILP-based heuristic, and is henceforth denoted by ILP-HEUR.

4 A Hybrid Algorithm Based on Solution Merging

As mentioned before, as a second way of taking profit from the ILP model presented in Section 3, we propose a hybrid algorithm based on solution merging. The pseudo-code of this algorithm, which is henceforth labelled HYB-ALG, is provided in Algorithm 1. Moreover, a corresponding flow diagram is provided in Figure 3. In the context of this algorithm, valid solutions to the problem are subsets of the complete set Z of variables which was introduced in the context of the ILP model. The meaning of a variable $z_{i,j}$ forming part of a solution S is simply that the corresponding assignment $a_{i,j}$ forms part of the corresponding common subsequence.

The stopping criterion for HYB-ALG is a CPU time limit. The main loop of the algorithm consists in the following actions. First, the best-so-far solution S_{bsf} is initialized either to the empty set ($\emptyset$), or to the solution produced by the heuristic from Section 2. This depends on the value of input parameter $h_{\text{opt}} \in \{\text{TRUE}, \text{FALSE}\}$, which is subject to parameter tuning. Then, at each iteration a subset S' of the complete set Z of variables is generated as follows. First, S' is initialized with the variables from the best-so-far solution S_{bsf}. Then, a number of n_{sols} solutions is probabilistically constructed in function GenerateRandomSolution(d_{rate}, l_{size}, Z) in line 6 of Algorithm 1. The variables that are found in these solutions are added to S'. Afterwards, optimal solution merging is applied to S', that is, an ILP solver is applied to find the best valid

solution that can be built from the variables in S' (see function ApplySolutionMerging($t_{\max}$, S') in line 9 of Algorithm 1). Hereby, parameter $t_{\max}$ is the CPU time limit for the ILP solver. This means that the output of this function might not be the optimal solution contained in S', but the best solution found within $t_{\max}$ seconds. Note that for applying an ILP solver to $S' \subseteq Z$, all the appearances of Z in the ILP model of Section 3 have to be replaced with S'. In case S'_{opt} is better than the current best-so-far solution S_{bsf}, solution S'_{opt} is stored as the new best-so-far solution (line 10). The output of the algorithm is the best-so-far solution S_{bsf}.

Algorithm 1 HYB-ALG for the LAPCS problem

1: **input:** strings x and y over alphabet Σ, values for parameters h_{opt}, n_{sols}, d_{rate}, l_{size}, and $t_{\max}$
2: **if** $h_{\text{opt}} =$ TRUE **then** $S_{\text{bsf}} :=$ ApplyHeuristic() **else** $S_{\text{bsf}} := \emptyset$ **end if**
3: **while** CPU time limit not reached **do**
4: **for** $i = 1, \ldots, n_{\text{sols}}$ **do**
5: $S' := S_{\text{bsf}}$
6: $S :=$ GenerateRandomSolution(d_{rate}, l_{size}, Z)
7: $S' := S' \cup S$
8: **end for**
9: $S'_{\text{opt}} :=$ ApplySolutionMerging($t_{\max}$, S')
10: **if** $|S'_{\text{opt}}| > |S_{\text{bsf}}|$ **then** $S_{\text{bsf}} := S'_{\text{opt}}$ **end if**
11: **end while**
12: **output:** S_{bsf}

The remaining algorithmic component, which is not yet described, is the probabilistic construction of solutions in function GenerateRandomSolution(d_{rate}, l_{size}, Z). The construction of a solution is done in two steps. First, a common subsequence of x and y is constructed without regarding the arc preservation constraints. Second, the *repair mechanism* described in Section 2 is used to transform S into a valid LAPCS solution.

The first step of the construction of a solution works as follows. The process starts with an empty solution $S = \emptyset$. Then, at each step the set of options $C \subseteq Z$ that may be added to S are generated as follows. Let p_x (respectively, p_y) be the index of the previously added variable concerning input string x (respectively, input string y). More specifically, if $z_{k,l}$ was the variable added to S in the previous construction step, then $p_x := k$ and $p_y := l$. In the case of the first construction step it holds that $p_x := 0$ and $p_y := 0$. The set of options C at each construction step contains, for each letter $a \in \Sigma$, the variable $z_{i,j} \in Z_a$ (if any) such that $p_x < i \leq k$ and $p_y < j \leq l$, for all $z_{k,l} \in Z_a$ with $k > p_x$ and $l > p_y$. Hereby, $Z_a \subseteq Z$ is the subset of variables which correspond to assignments concerning letter a, that is, for each $z_{i,j} \in Z_a$ it holds that $x[i] = y[j] = a$.

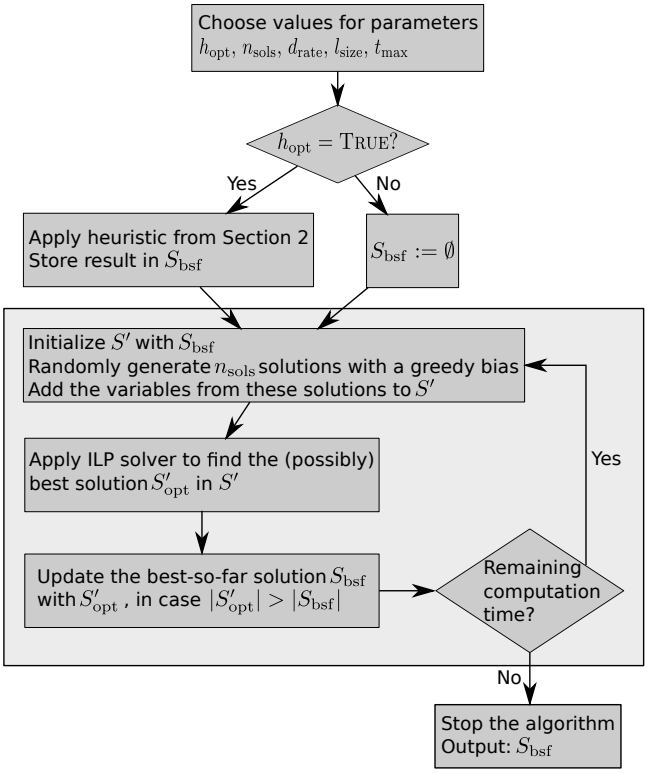


Figure 3: Flow diagram of HYB-ALG (see also Algorithm 1).

Moreover, the following weighting function assigns a weight to each option:

$$w(z_{i,j}) := \frac{i - p_x}{l_x - p_x} + \frac{j - p_y}{l_y - p_y} \quad \forall\ z_{i,j} \in C \tag{6}$$

Note that this is a known greedy function for longest common subsequence problems (see, for example, [29, 30]). At each construction step, exactly one variable is chosen from C and added to S. For doing so, first, a value $rand$ is chosen uniformly at random from $[0, 1]$. In case $rand \leq d_{\text{rate}}$, where d_{rate} is a parameter of the algorithm, the variable $z_{i,j} \in C$ with the smallest weight value is deterministically chosen. Otherwise, a candidate list $L \subseteq C$ of size $\min\{l_{\text{size}}, |C|\}$ containing the options with the lowest weight values is generated and exactly one variable $z_{i,j} \in L$ is then chosen uniformly at random and added to S. Note that l_{size} is another parameter of the solution construction process. The solution construction is finished when the set of options is empty. Afterwards, the repair mechanism is applied as outlined above.

5 Experimental Evaluation

In the following we describe the experimental evaluation that we conducted, considering both techniques introduced in this work (ILP-HEUR and HYB-ALG), in comparison to the heuristic from the literature (HEURISTIC). Note that HEURISTIC was re-implemented using the same data structures as HYB-ALG. All algorithms were implemented in ANSI C++ using GCC 4.6.3. In addition, the ILP models involved in the three techniques were solved with the ILP solver IBM ILOG CPLEX v12.6 in one-threaded mode. The experimental evaluation has been performed on a cluster of PCs with Intel(R) Xeon(R) CPU 5670 CPUs of 12 nuclei of 2933 MHz and at least 40 Gigabytes of RAM. Note that we also tried to apply CPLEX to the complete ILP models for each problem instance. However, the models were too large, even in the case of the smallest problem instances.

The remainder of this section is organized as follows. First, the set of benchmark instances—consisting of artificial and real instances—is described. Second, ILP-HEUR is experimentally evaluated. Third, after describing the tuning experiments conducted in order to find well-working parameter values for HYB-ALG, we compare the best results of ILP-HEUR and the results of HYB-ALG with those of HEURISTIC. Finally, the impact of the optimal solution merging component on HYB-ALG is studied.

5.1 Benchmark Instances

Both artificial and real benchmark instances consisting of RNA sequences were considered. The first set, labelled SET1, consists of artificial problem instances. Each of these instances consists of two artificially generated RNA strings of length $l_x = l_y \in \{100, 200, \ldots, 900, 1000\}$. The probability of each letter and each position was chosen to be 1/4. Moreover, for each input string we randomly generated a number of $n_{\text{arcs}} \in \{l_x/10, l_x/5, l_x/2\}$ randomly generated arcs. Hereby, it was taken care that all n_{arcs} arcs were different. For each combination of l_x and n_{arcs} we randomly generated 30 problem instances. This makes a total of 900 problem instances.

The second benchmark set, labelled SET2, consists of arc-annotated RNA sequences from the RNase P Database [31]. In total we assembled 10 problem instances, whose characteristics are described in Table 3. The secondary structures of the RNA sequences involved in instances Real_1 and Real_8 are exemplary shown in Figure 10, at the end of this paper.

5.2 Experimental Study of ILP-HEUR

After an initial experimentation, it was decided to apply ILP-HEUR to all problem instances with eight different values for the separation factor (p). More specifically, CPLEX was used to solve $\text{ILP}^r(p)$ for $p \in \{1, 2, 3, 4, 5, 10, 15, 20\}$ and with a computation time limit of l_x CPU seconds in the context of the artificial problem instances (remember that $l_x = l_y$ is the length of the two input

Table 3: Characteristics of the real-life instances. All 20 RNA sequences, together with their secondary structure, were downloaded from the RNase P Database [31].

Instance	First String			Second string		
name	RNA	Lenght	Arcs	RNA	Lenght	Arcs
Real_1	*Allochromatium vinosum*	369	119	*Haemophilus influenza*	377	124
Real_2	*Bacteroides thetaiotaomicron*	361	121	*Porphyromonas gingivalis*	398	131
Real_3	*Halococcus morrhuae*	475	154	*Haloferax volcanii*	433	142
Real_4	*Klebsiella pneumoniae*	383	127	*Escherichia coli*	377	124
Real_5	*Methanococcus jannaschii*	252	75	*Archaeoglobus fulgidus*	229	67
Real_6	*Methanosarcina barkeri*	371	115	*Pyrococcus abyssi*	330	100
Real_7	*Mycoplasma genitalium*	384	119	*Mycoplasma pneumoniae*	369	112
Real_8	*Saccharomyces kluveri*	336	90	*Schizosaccharomyces octosporus*	281	71
Real_9	*Serratia marcescens*	378	125	*Shewanella putrefaciens*	354	115
Real_10	*Streptomyces bikiniensis*	398	135	*Streptomyces lividans*	405	138

sequences). The same values for p and a computation time limit of 300 CPU seconds was used for the application to the real problem instances.

In order to be able to interpret the results, the information is plotted in Figure 4 (concerning the artificial instances from SET1) and in Figure 5 (concerning the real instances from SET2).[3] However, remember that in the case of the artificial instances, results are shown in terms of averages over 30 problem instances of the same type, whereas in the case of the real instances, results are shown for each single instance. The plots, whose y-axis ranges over the eight values for p, show the following information. The obtained solution quality is shown by the bars. The exact solution qualities are printed at the top of the bars. ILP$^r(4)$, for example, obtained an average solution quality of 55.7 in case of the artificial problem instances with $l_x = 100$ and $n_{\text{arcs}} = l_x/10$ arcs (see Figure 4a). An additional information is shown by the curves in each plot. These curves show the size of the model used in ILP$^r(p)$, first, in terms of the percentage of the variables of the original model that are used in the reduced model (y-axis), and, second, in terms of the absolute number of variables in the reduced model (see the numbers above the line-points). The reduced model considered in ILP$^r(3)$, for example, uses about 7% (absolute value: 7415 variables) in the case of the instances with $l_x = 700$ and $n_{\text{arcs}} = l_x/2$ arcs (see Figure 4f).

The following observations can be made. As expected, the results obtained by applying CPLEX to ILP$^r(p)$ improve with a growing value of p.[4] However, starting from a certain value of p, the results rapidly become worse. This is because starting from this value of p, solving the reduced model with CPLEX is not efficient due to the size even of the reduced model. For example, in the case of instances with $l_x = 400$ and $n_{\text{arcs}} = l_x/10$ arcs, the obtained solution quality starts to degrade with $p = 5$. Obviously, with a growing length of the input

[3]Note that for space reasons we only provide the plots for artificial instances with $l_x \in \{100, 400, 700, 1000\}$ and for real instances Real_1, Real_4, Real_7 and Real_10. The best results of ILP$^r(p)$ are provided for all instances in the result tables of Section 5.3.

[4]Note, in this context, that with $p = 100$, ILP$^r(p)$ refers to the original model, that is, a value of $p = 100$ does not cause any model reduction.

strings, this degradation of the results starts with increasingly lower values of p. In fact, when considering instances with $l_x = 1000$, the only reduced model that can be solved is the one with $p = 1$.

On the basis of the obtained results, the following recommendation can be made for the choice of a suitable value for p when faced with a new, unknown, problem instance. The graphics in Figures 4 and 5 show that the employed ILP solver's performance generally starts to degrade for models of approximately 3000 variables. Therefore, we recommend to choose p such that the resulting model size approaches 3000 variables as far as possible from below. However, this recommendation obviously depends on the utilized computing platform.

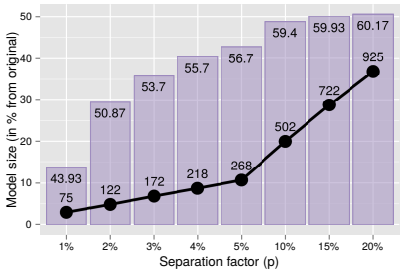
5.3 Numerical Comparison

In the following we first describe the tuning procedure applied for finding well-working parameter values for Hyb-Alg. Then we present the numerical comparison of the results obtaind by Ilp-Heur and Hyb-Alg with Heuristic from the literature.

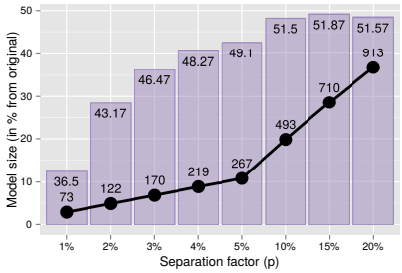
5.3.1 Tuning Procedure and Sensitivity Analysis

The automatic configuration tool irace [32] was used for tuning the parameters of Hyb-Alg. The following parameters of Hyb-Alg were considered for tuning: (h_{opt}) concerning the decision whether to initialize the best-found solution, (n_{sols}) the number of solution constructions per iteration, (d_{rate}) the determinism rate, (l_{size}) the candidate list size, ($t_{\max}$) the maximum time in seconds allowed for solution merging (at each call of the solution merging procedure). In particular, Hyb-Alg was tuned separately for each input string length, which—after initial experiments—seemed to have a greater influence on the behavior of the algorithm than the number of arcs. For each $l_x \in \{100, 200, \ldots, 900, 1000\}$ we randomly generated two tuning instances for each of the three values of n_{arcs}. This makes a total of six tuning instances for each considered value of l_x. The tuning process for each l_x was given a budget of 1000 runs of Hyb-Alg, where each run was given a computation time limit of l_x CPU seconds. Finally, the following parameter value ranges were considered concerning the five parameters of Hyb-Alg:

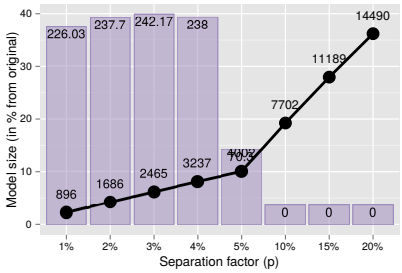
- $h_{\text{opt}} \in \{\text{True}, \text{False}\}$, where False indicates no initialization, whereas True indicates the initialization with the solution obtained by Heuristic.
- $n_{\text{sols}} \in \{5, 10, 20\}$
- $d_{\text{rate}} \in \{0.0, 0.3, 0.5, 0.7, 0.9\}$, where a value of 0.0 means that the selection of the assignment to be added to the partial solution under construction is always done randomly from the candidate list, while a value of 0.9 means that solution constructions are nearly deterministic.



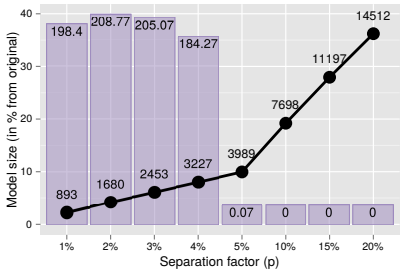
(a) $l_x = 100$, $n_{\text{arcs}} = l_x/10$.



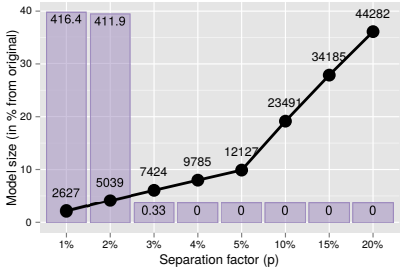
(b) $l_x = 100$, $n_{\text{arcs}} = l_x/2$.



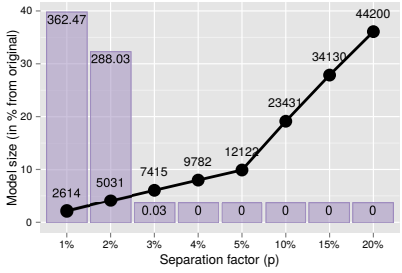
(c) $l_x = 400$, $n_{\text{arcs}} = l_x/10$.



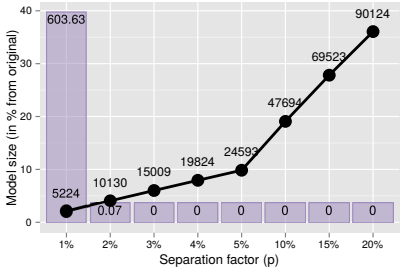
(d) $l_x = 400$, $n_{\text{arcs}} = l_x/2$.



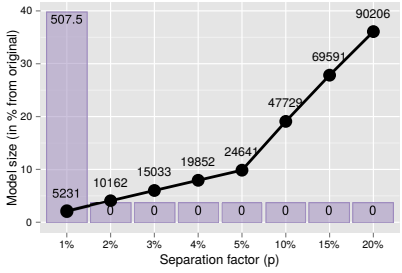
(e) $l_x = 700$, $n_{\text{arcs}} = l_x/10$.



(f) $l_x = 700$, $n_{\text{arcs}} = l_x/2$.

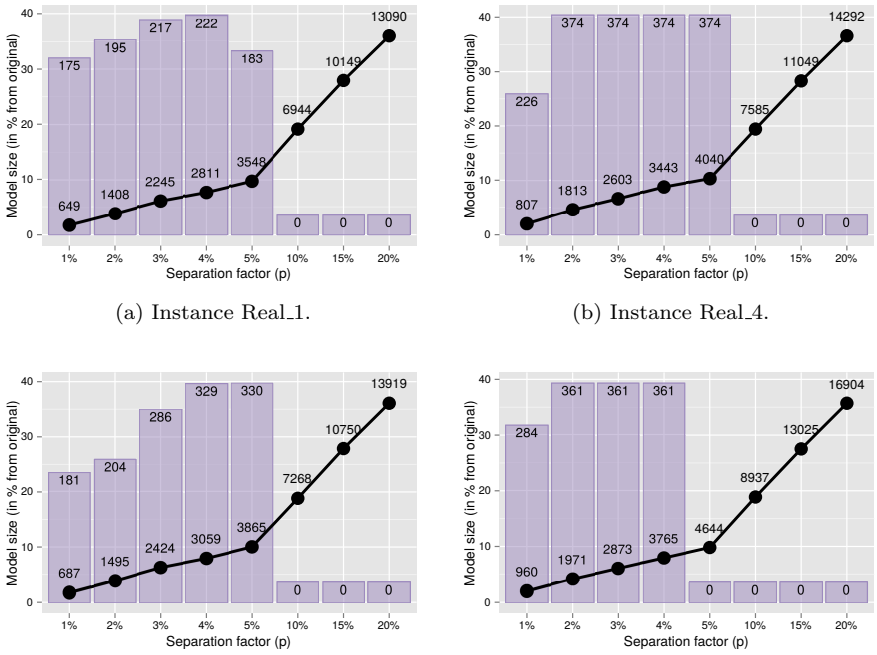


(g) $l_x = 1000$, $n_{\text{arcs}} = l_x/10$.



(h) $l_x = 1000$, $n_{\text{arcs}} = l_x/2$.

Figure 4: Results of applying CPLEX to solve the reduced models $\text{ILP}^r(p)$ for the artificial instances from SET1 with $l_x \in \{100, 400, 700, 1000\}$. The x-axis ranges over the considered values for p. The structure of the plots is outlined in the text.



(a) Instance Real_1.

(b) Instance Real_4.

(c) Instance Real_7.

(d) Instance Real_10.

Figure 5: Results of applying CPLEX to solve the reduced models $\text{ILP}^r(p)$ for the real instances Real_1, Real_4, Real_7 and Real_10 from SET2. The x-axis ranges over the considered values for p. The structure of the plots is outlined in the text.

- $l_{\text{size}} \in \{1, 2, 3, 4\}$
- $t_{\max} \in \{1.0, 5.0, 10.0, 20.0\}$ (in seconds).

The tuning runs with irace produced the configurations of HYB-ALG as shown in Table 4. The following trends can be observed. First of all, the number of solution constructions per iteration is always generally set to 10 for smaller problem instances and to five for larger problem instances. This is because the smaller n_{sols}, the smaller is the ILP model that has to be solved by CPLEX in the context of solution merging at each iteration of the algorithm. Moreover, the smaller the ILP model, the more efficient is CPLEX in solving such a model. The values of d_{rate} are consistently between 0.5 and 0.7 (and most of the times at 0.5), whereas the values of l_{size} are consistently set to two or three. The settings of $t_{\max}$ are rather difficult to interpret. The rather high time limits for some of the instances size ($l_x \in \{300, 400, 700\}$) might be due to the fact that the involved ILP models are solved rather quickly such that the time limit does not play a role. Finally, in all cases the tuning procedure identified the

Table 4: Results of tuning HYB-ALG with irace.

l_x	h_{opt}	n_{sols}	d_{rate}	l_{size}	$t_{\max}$
100	TRUE	10	0.7	3	1.0
200	TRUE	5	0.5	3	10.0
300	TRUE	10	0.5	2	20.0
400	TRUE	10	0.7	3	20.0
500	TRUE	10	0.5	2	10.0
600	TRUE	5	0.5	3	10.0
700	TRUE	5	0.7	2	20.0
800	TRUE	5	0.5	2	5.0
900	TRUE	5	0.7	3	5.0
1000	TRUE	5	0.5	2	5.0

usefulness of the initialization of the best-found solutions.

In order to study the sensitivity of the algorithm concerning the chosen parameter setting, the following experiments were performed. First, HYB-ALG was applied exactly once—using the parameter values determined by irace—to each instance of SET1 (see Section 5.1). The time limit for each run was l_x seconds, the same as for the tuning procedure. The same experiments were repeated four times, using the same parameter values, apart from one change for each of the four repetitions: (1) $n_{\text{sols}} = 2$ (very low number of solution constructions per iteration), (2) $n_{\text{sols}} = 20$ (rather high number of solution constructions per iteration), (3) $d_{\text{rate}} = 0.0$ (lowest possible determism rate value), and (4) $d_{\text{rate}} = 0.9$ (the highest considered determism rate value during tuning). In the following we will refer to these four algorithm configurations as the *extreme algorithm configurations*. Note that among the four parameters of HYB-ALG, n_{sols} and d_{rate} were chosen for the following reason. According to our experience, generally n_{sols} has a rather high impact on the algorithm performance, while the opposite is the case for $t_{\max}$ (which is simply a parameter to avoid wasting valuable computation time). Moreover, d_{rate} and l_{size} are highly correlated parameters. Therefore, only one of them (d_{rate}) was chosen.

For each of the five tested algorithm configurations (the standard one plus four extreme configurations), the obtained average solution quality was calculated for the 30 instances for each combination of l_x and n_{arcs}. In Figure 6, the results are shown in terms of the average improvement (in percent) of the algorithm configuration obtained by tuning over the four extreme algorithm configurations. The following observations can be made. First, the algorithm seems most sensitive against an increase of the number of solutions probabilistically generated per iteration (see the box with label $n_{\text{sols}} = 20$). On the other side, reducing this number to $n_{\text{sols}} = 2$ does not seem to have much of an effect. Finally, increasing d_{rate} to 0.9 or reducing its value to 0.0 has a noticable effect, however, not comparable to increasing the number of solution constructions.

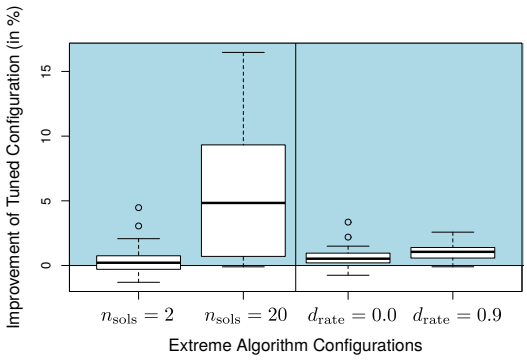


Figure 6: Improvement of Hyb-Alg over the four extreme algorithm configurations (in percent). Each box shows the average differences for the 30 instances for each combination of l_x and n_{arcs}.

5.3.2 Results

All techniques were applied exactly once to each problem instance. The computation time limit used for Ilp-Heur and Hyb-Alg was l_x CPU seconds. Note that stopping the algorithms using the same computation time limit is the only feasible option, because both Ilp-Heur and Hyb-Alg make internal use of a black-box ILP solver. For the computation time used by this solver it is simply unknown how many solutions are evaluated. Obviously, using a CPU time limit as stopping criterion has the disadvantage that future competitors must re-run our algorithms on their hardware.[5] In order to faciliate this, we offer both the problem instances and executables of both Ilp-Heur and Hyb-Alg at `https://www.iiia.csic.es/~christian.blum/research.html`. Moreover, note that the CPU time limit is only checked once at the end of each iteration of Hyb-Alg. In order to avoid to consider solutions that are possibly found (slightly) over the CPU time limit, whenever a new best solution is found, the exact current time is determined, and in case this time is over the CPU time limit, the corresponding solution is not recorded.

The results concerning the artificial problem instances from Set1 are presented in Table 5. Each row provides the results of Heuristic, Ilp-Heur and Hyb-Alg in terms of the average solution quality obtained for the 30 problem instances of the corresponding combination of l_x and n_{arcs}. Note that in the case of Ilp-Heur, for each combination of l_x and n_{arcs} the results are the ones with the best-performing separation factor (p). The value of p is indicated in the second column correponding to the results of Ilp-Heur. Moreover, the third column for Ilp-Heur shows the corresponding model size in terms of the number of variables. The column with heading **time** shows the average computation time for the 30 problem instances in the case of Heuristic, and the

[5] For more information on this issue see, for example, [33].

Table 5: Experimental results concerning the artificial problem instances from Set1.

n	# arcs	Heuristic		Ilp-Heur			Hyb-Alg	
		result	**time**	**result**	p	# vars	**result**	**time**
100	10	55.73	< 1	60.17	20	925.37	59.67★	11.43
	20	51.63	< 1	58.13	20	920.97	57.50★	11.63
	50	42.63	< 1	51.87	15	709.60	50.73	16.66
200	20	113.77	< 1	120.40★	5	1020.23	121.70	38.28
	40	104.13	< 1	115.77★	5	1025.87	116.70	61.36
	100	87.10	< 1	104.57	5	1017.87	103.73	76.77
300	30	170.43	< 1	181.30	5	2249.63	181.07★	70.01
	60	156.87	< 1	174.97	4	1840.33	173.93★	90.62
	150	132.40	< 1	157.13	3	1404.63	154.70	155.99
400	40	229.20	< 1	242.17★	3	2465.00	242.70	191.72
	80	210.93	< 1	233.23	3	2463.13	231.47★	233.49
	200	175.63	< 1	208.77	2	1680.47	205.40	289.37
500	50	289.20	< 1	300.63★	2	2597.53	302.27	250.46
	100	263.80	< 1	291.23	2	2587.97	289.83★	298.96
	250	220.43	< 1	259.50	2	2594.77	257.33	356.59
600	60	346.63	< 1	364.13★	2	3707.07	366.03	324.61
	120	317.93	< 1	350.97	2	3709.03	347.57★	407.48
	300	266.03	< 1	309.20	1	1933.03	304.60★	420.62
700	70	404.53	< 1	416.40★	1	2626.83	418.40	372.52
	140	369.97	< 1	400.60	1	2608.07	398.63★	423.64
	350	308.57	< 1	362.74	1	2613.83	359.67★	461.63
800	80	461.17	< 1	476.63	1	3374.97	484.43	420.71
	160	424.57	< 1	462.07★	1	2390.41	462.60	572.70
	400	353.57	< 1	414.33	1	3380.53	411.80★	470.47
900	90	520.87	< 1	539.37★	1	4246.93	542.07	516.09
	180	477.90	< 1	522.40	1	4241.97	519.10	560.27
	450	398.97	< 1	463.27	1	4244.20	462.20★	682.07
1000	100	578.30	< 1	603.63★	1	5223.50	605.10	535.42
	200	531.33	< 1	583.30	1	5223.57	577.67	659.28
	500	443.33	< 1	507.50	1	5231.23	514.80	664.53

average time at which the best solution of a run was found in the case of Hyb-Alg. The best result of each table row is marked by a lightgrey background. In addition, we applied a statistical significance test to the results of each table row. More specifically, in each table row all approaches were compared to the best-performing approach and the results of those approaches who are statistically equivalent to the best-performing approach are marked by the ★ symbol (significance level of 0.05). The statistical differences have been assessed using the Friedman test and the p-values have been corrected for multiple comparison using Finner's procedure [34]. All the tests were performed using `R`'s **scmamp** package [35], available at `https://github.com/b0rxa/scmamp`.

The following observations can be made:

- HEURISTIC (which is a deterministic approach) is very fast. Its application to any of the problem instances requires less than one CPU second. On the other side, the results of HEURISTIC are always the worst ones in the comparison.

- ILP-HEUR is generally the best-performing technique for instances with rather many arcs, that is, $n_{\text{arcs}} \in \{n/5, n/2\}$. Concerning the instances with the fewest number of arcs—that is, $n_{\text{arcs}} = n/10$—HYB-ALG seems to have slight advantages over ILP-HEUR, especially with growing instance size.

- Both ILP-HEUR and HYB-ALG clearly require more computation time than the simple deterministic approach HEURISTIC. However, the LAPCS problem does, generally, not require specially low running time limits. Therefore, longer running times are not really problematic.

In order to study the impact of the optimal solution merging operator of HYB-ALG we applied HYB-ALG without the application of this component, with the same parameter values and the same run time limits. The resulting algorithm boils down to a multi-start heuristic that keeps constructing solutions probabilistically over and over again, until the CPU time limit is reached. This algorithm is henceforth called Ms-Heur. In order to study the magnitude of the improvement of HYB-ALG over MS-HEUR, we show the percentage improvement of HYB-ALG over MS-HEUR—by means of boxplots—for each combination of l_x and n_{arcs} in the three graphics of Figure 7. The x-axis of each graphic ranges from $l_x = 100$ to $l_x = 1000$. These graphics show, first, that the improvement of HYB-ALG over MS-HEUR is generally around 20%. Moreover, it can be observed that the improvements become slightly bigger with a growing number of arcs. In summary, this means that the solution merging component is an essential part of HYB-ALG.

In order to confirm the findings described so far, we aimed for detecting statistical differences between the algorithms (if any) for subsets of the considered instances.[6] For doing so, all the algorithms have been compared simultaneously using Friedman's test. Then, given that in all the cases the test rejected the hypothesis that all the algorithms perform equally, all the pairwise comparisons have been performed using the Nemenyi post-hoc test [34]. The corresponding results are shown in Figure 8 by means of so-called criticial difference plots. Briefly, each algorithm is positioned in the segment according to its average ranking concerning the considered subset of instances. Then, the critical difference (CD) is computed for a significance level of 0.05 and the performance of those algorithms that have a difference lower than CD are regarded as equal—that is, no difference of statistical significance can be detected. This is indicated in the graphics by horizontal bars joining the respective algorithms. However, no such case occurs in the presented graphics. In summary, the graphics confirm

[6] Again, all the tests and the plots were generated using R's **scmamp** package [35].

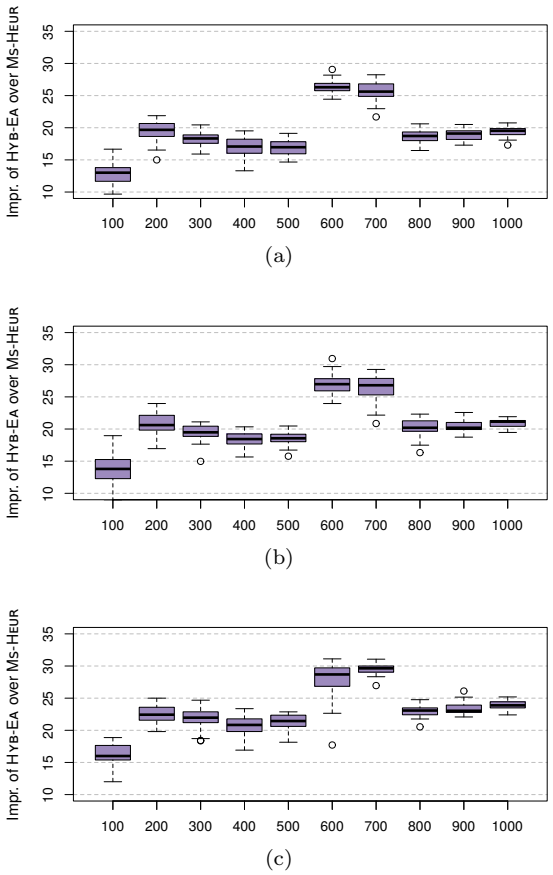
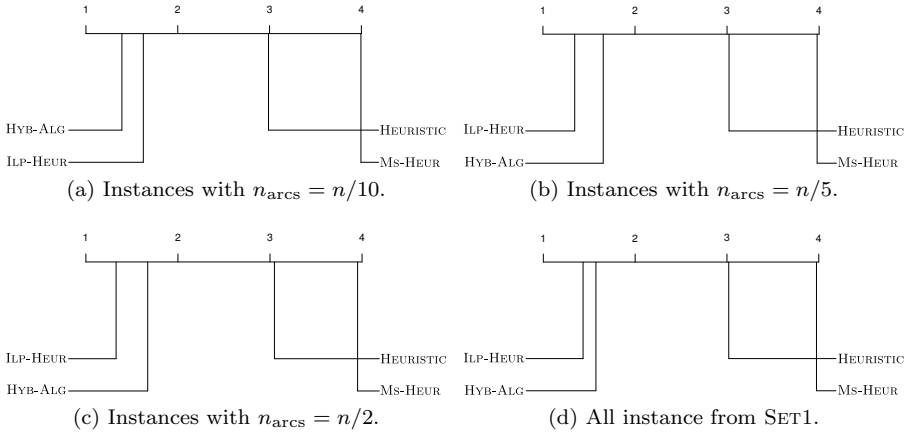


Figure 7: Improvement of Hyb-Alg over Ms-Heur (in percent). Each box shows the differences for the corresponding 30 instances of a specific input string length $l_x \in \{100, \ldots, 1000\}$. (a) Instances with $n_{\text{arcs}} = l_x/10$, (b) instances with $n_{\text{arcs}} = l_x/5$, (c) instances with $n_{\text{arcs}} = l_x/2$.

the findings described above.

In the last set of experiments we applied the three algorithms considered in this work to the set of 10 real problem instances (Set2). The results are provided in Table 6, in the following way. Heuristic, due to being a deterministic approach, was applied exactly once to each problem instance. The corresponding result can be found in columns with headings **result** and **time**. The results of Ilp-Heur correspond, again, to the ones obtained with the best-working value of p. Remember that, for each problem instance, eight values of p were tested with a computation time limit of 300 CPU seconds per application. Hyb-Alg was applied 30 times to each problem instance, with the same computation time limit of 300 CPU seconds per application. We provide the



(a) Instances with $n_{\text{arcs}} = n/10$.

(b) Instances with $n_{\text{arcs}} = n/5$.

(c) Instances with $n_{\text{arcs}} = n/2$.

(d) All instance from Set1.

Figure 8: Criticial difference plots for subsets of Set1 (see (a) to (c)), and globally for all instances of Set1 (see (d)). The axis shows the average ranking of the algorithms concerning the considered subset of instances. Horizontal bars connect algorithms that are statistically equivalent (however, no such case exists).

Table 6: Experimental results concerning the real problem instances from Set2.

inst.	Heuristic		Ilp-Heur	Hyb-Alg		
	result	**time**	**result**	**best**	**avg.**	**time**
Real_1	238	< 1	222	268	267.10	80.92
Real_2	260	< 1	214	291	289.40	84.93
Real_3	265	< 1	242	294	292.40	171.78
Real_4	373	< 1	374	374	373.80	0.02
Real_5	152	< 1	138	178	177.30	58.96
Real_6	183	< 1	188	209	208.60	131.35
Real_7	316	< 1	330	330	330.00	7.40
Real_8	153	< 1	163	177	174.80	174.40
Real_9	285	< 1	198	302	301.10	24.89
Real_10	343	< 1	361	359	359.00	11.38

best result obtained over 30 applications (column **best**), the average of the 30 results obtained (column **avg.**), and the average time at which the best solution of each run was found (column **time**). Again, the best result for each of the 10 instances is marked by a lightgrey background. Surprisingly, the results of Ilp-Heur (apart from Real_4, Real_7 and Real_10) are now clearly worse than those of Hyb-Alg. They are often even worse than those of Heuristic.

In order to study the reasons for this behaviour we plotted histograms of the *percentage gaps* concerning the variables (respectively, assignments) found in solutions generated by Hyb-Alg. The percentage gap concerning a variable

$z_{i,j}$ that is contained in a solution of Hyb-Alg is calculated as follows:

$$p_{\text{gap}} := \left\lfloor \frac{|i-j|}{l} \cdot 100 \right\rfloor \tag{7}$$

Note, in this context, that a reduced ILP model $\text{ILP}^r(p)$ with maximum separation factor p as defined in Section 3.1 only contains the variables with a maximum percentage gap of p. The histograms provided in Figure 9 show, for each value of p_{gap}, the number of variables from the corresponding solution(s) that have this p_{gap} value. The graphics in Figures 9a–9d show the histograms concerning the 30 Hyb-Alg solutions for representative cases of artificial problem instances. It can be observed that most variables have a p_{gap} value of zero. Additionally, with inreasing value of p_{gap}, the number of variables decreases. Furthermore, remember that CPLEX was even able to find very good solutions—often better ones than Hyb-Alg—with a limit of $p = 1$. On the other side, the histograms concerning the real instances look very different. Consider, for example, the histogram concerning the Hyb-Alg solutions—that is, the 10 best solutions out of 10 runs—for real instance Real_1 in Figure 9e. These solutions contain many variables with a p_{gap} value of two, six, and nine. This means that the solution in this case is often a string of which (1) large pieces can be found nearly consecutively in both input strings, and (2) these pieces occur not at the same positions in the two input sequences. On the contrary, there might be a rather large gap between the matched pieces from both input sequences. As a consquence, in the case of real instances, the p-value for the reduced model $\text{ILP}^r(p)$ must be rather high in order to obtain good solutions. In many cases, the value of p must be so high that the corresponding reduced model can not efficiently be solved anymore by CPLEX. This is why the results of Ilp-Heur are so much worse than the ones of the competitor algorithms in the case of real instances.

6 Conclusion

This paper has dealt with an NP-hard combinatorial problem from the bioinformatics field: the so-called longest arc-preserving common subsequence problem. First, we presented an ILP model for the problem. As it resulted inviable to apply general purpose ILP solvers to solve this model, we studied different ways of making use of the model within heuristics. The first one consisted in reducing the model in a sensible way, and subsequently applying a general purpose ILP solver to the reduced model. The second one was a simple hybrid algorithm that uses the model within an optimal solution merging operator. The results have shown that the first option often outperforms the second one in the context of artificially generated problem instances. However, in the context of real problem instances, the proposed hybrid algorithm was the better option.

In future work we will try to replace the probabilistic way of constructing solutions by a probabilistic version of the Smith and Waterman algorithm. Moreover, we will try to incorporate already information about the arcs into

the solution construction procedure. In this way we might be able to construct longer arc-preserving common subsequences than with the current procedure, which first constructs a solution without regarding the arc information, and subsequently applies a repair procedure.

Acknowledgment

This work was partially funded by project SGR 2014-1034 (AGAUR, Generalitat de Catalunya). Our experiments have been executed in the High Performance Computing environment managed by the RDlab at the Technical University of Barcelona (`http://rdlab.cs.upc.edu`) and we would like to thank them for their support.

References

[1] T. Jiang, G. Lin, B. Ma, K. Zhang, The longest common subsequence problem for arc-annotated sequences, Journal of Discrete Algorithms 2 (2) (2004) 257–270.

[2] P. A. Evans, Finding common subsequences with arcs and pseudoknots, in: M. Crochemore, M. Paterson (Eds.), Proceedings of CPM 1999 – 10th Annual Symposium on Combinatorial Pattern Matching, Vol. 1645 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1999, pp. 270–280.

[3] J. K. H. Chiu, Y.-P. P. Chen, A comprehensive study of RNA secondary structure alignment algorithms, Briefings in Bioinformatics. In press. URL `https://dx.doi.org/10.1093/bib/bbw009`

[4] C. S. Iliopoulos, M. Sohel Rahman, A new efficient algorithm for computing the longest common subsequence, Theory of Computing Systems 45 (2) (2009) 355–371.

[5] M. Castelli, S. Beretta, L. Vanneschi, A hybrid genetic algorithm for the repetition free longest common subsequence problem, Operations Research Letters 41 (6) (2013) 644–649.

[6] D. Maier, The complexity of some problems on subsequences and supersequences, Journal of the ACM 25 (1978) 322–336.

[7] C. Blum, M. J. Blesa, M. López-Ibáñez, Beam search for the longest common subsequence problem, Computers & Operations Research 36 (12) (2009) 3178–3186.

[8] S. R. Mousavi, F. Tabataba, An improved algorithm for the longest common subsequence problem, Computers & Operations Research 39 (3) (2012) 512–520.

[9] P. A. Evans, Algorithms and complexity for annotated sequence analysis, Ph.D. thesis, University of Victoria (1999).

[10] T. Jiang, G. Lin, B. Ma, K. Zhang, A general edit distance between RNA structures, Journal of Computational Biology 9 (2) (2002) 371–388.

[11] T. F. Smith, M. S. Waterman, Identification of common molecular subsequences, Journal of Molecular Biology 147 (1) (1981) 195–197.

[12] G. Blin, Combinatorial objects in bio-algorithmics: Related problems and complexities, Ph.D. thesis, Université Paris-Est (2012).

[13] G. Lin, Z.-Z. Chen, T. Jiang, J. Wen, The longest common subsequence problem for sequences with nested arc annotations, Journal of Computer and System Sciences 65 (3) (2002) 465–480.

[14] G. Blin, S. Hamel, S. Vialette, Comparing RNA structures with biologically relevant operations cannot be done without strong combinatorial restrictions, in: M. Rahman, S. Fujita (Eds.), Proceedings of WALCOM 2010 – International Workshop on Algorithms and Computation, Vol. 5942 of Lecture Notes in Computer Science, Springer, 2010, pp. 149–160.

[15] T. Jiang, G.-H. Lin, B. Ma, K. Zhang, The longest common subsequence problem for arc-annotated sequences, in: R. Giancarlo, D. Sankoff (Eds.), Proceedings of PCM 2000 – Annual Symposium on Combinatorial Pattern Matching, Vol. 1848 of Lecture Notes in Computer Science, Springer, 2000, pp. 154–165.

[16] C. Blum, M. J. Blesa, A hybrid evolutionary algorithm based on solution merging for the longest arc-preserving common subsequence problem, in: Proceedings of CEC 2017 – IEEE Congress on Evolutionary Computation, IEEE press, 2017, pp. 129–136.

[17] G. L. Nemhauser, L. A. Wolsey, Integer and Combinatorial Optimization, Wiley & Sons, 1988.

[18] C. Blum, P. Festa, Metaheuristics for String Problems in Bio-informatics, John Wiley & Sons, 2016.

[19] C. Aggarwal, J. Orlin, R. Tai, Optimized crossover for the independent set problem, Operations Research 45 (1997) 226–234.

[20] R. Ahuja, J. Orlin, A. Tiwari, A greedy genetic algorithm for the quadratic assignment problem, Computers & Operations Research 27 (2000) 917–934.

[21] C. Blum, A new hybrid evolutionary algorithm for the k-cardinality tree problem, in: M. Cattolico et al. (Ed.), Proceedings of GECCO 2006 – Genetic and Evolutionary Computation Conference, ACM Press, 2006, pp. 515–522.

[22] P. Borisovsky, A. Dolgui, A. Eremeev, Genetic algorithms for a supply management problem: MIP-recombination vs. greedy decoder, European Journal of Operational Research 195 (3) (2009) 770–779.

[23] A. V. Eremeev, J. V. Kovalenko, Experimental evaluation of two approaches to optimal recombination for permutation problems, in: F. Chicano, B. Hu, P. García-Sánchez (Eds.), Proceedings of EvoCOP 2016 – 16th European Conference on Evolutionary Computation in Combinatorial Optimization, Vol. 9595 of Lecture Notes in Computer Science, Springer International Publishing, 2016, pp. 138–153.

[24] A. V. Eremeev, On complexity of optimal recombination for binary representations of solutions, Evolutionary Computation 16 (1) (2008) 127–147.

[25] A. V. Eremeev, J. V. Kovalenko, Optimal recombination in genetic algorithms for combinatorial optimization problems: Part I, Yugoslav Journal of Operations Research 24 (1) (2014) 1–20.

[26] A. V. Eremeev, J. V. Kovalenko, Optimal recombination in genetic algorithms for combinatorial optimization problems: Part II, Yugoslav Journal of Operations Research 24 (2) (2014) 165–186.

[27] D. Pisinger, S. Ropke, Large neighborhood search, in: M. Gendreau, J. Y. Potvin (Eds.), Handbook of Metaheuristics, 2nd Edition, Vol. 146 of International Series in Operations Research & Management Science, Springer, 2010, pp. 399–419.

[28] R. E. Tarjan, A. E. Trojanowski, Finding a maximum independent set, SIAM Journal on Computing 6 (3) (1977) 537–546.

[29] C. B. Fraser, Subsequences and supersequences of strings, Ph.D. thesis, University of Glasgow (1995).

[30] K. Huang, C. Yang, K. Tseng, Fast algorithms for finding the common subsequences of multiple sequences, in: Proceedings of the 2004 International Computer Symposium, IEEE press, 2004, pp. 1006–1011.

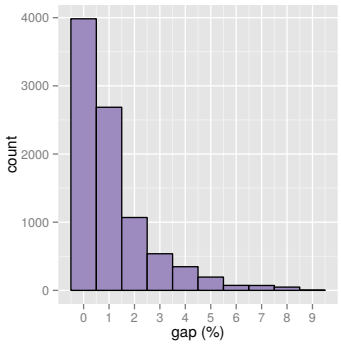
[31] J. W. Brown, The ribonuclease P database, Nucleic Acids Research 27 (1) (1999) 314–314.

[32] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43 – 58.

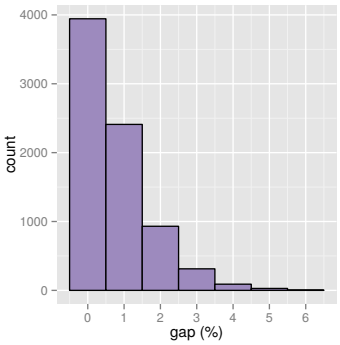
[33] M. Črepinšek, S.-H. Liu, M. Mernik, Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them, Applied Soft Computing 19 (2014) 161–170.

[34] S. García, F. Herrera, An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons, Journal of Machine Learning Research 9 (2008) 2677 – 2694.

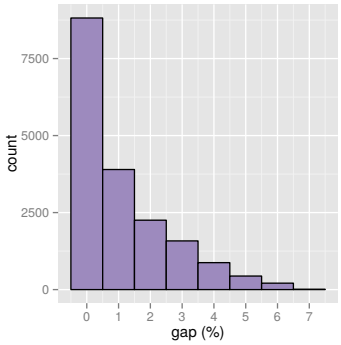
[35] B. Calvo, G. Santafe, scmamp: Statistical comparison of multiple algorithms in multiple problems, The R Journal 8 (1) (2016) 248–256.



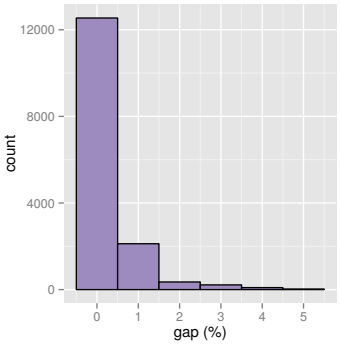
(a) Artificial instances with $l_x = 500$ and $n_{\text{arcs}} = n/10$.



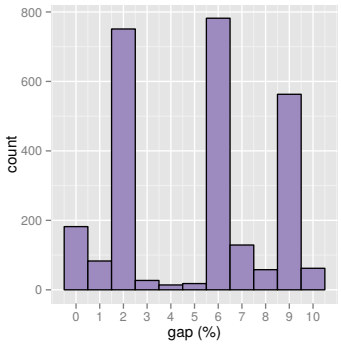
(b) Artificial instances with $l_x = 500$ and $n_{\text{arcs}} = n/2$.



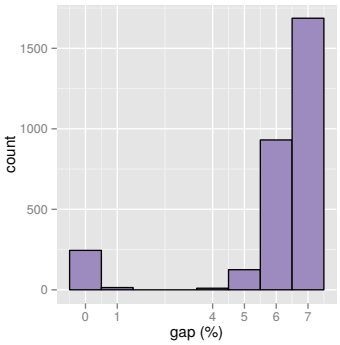
(c) Artificial instances with $l_x = 1000$ and $n_{\text{arcs}} = n/10$.



(d) Artificial instances with $l_x = 1000$ and $n_{\text{arcs}} = n/2$.



(e) Real instance Real_1.



(f) Real instance Real_9

Figure 9: Histograms showing the distribution of the p_{gap} values concerning the solutions produced by Hyb-Alg for representative instances.

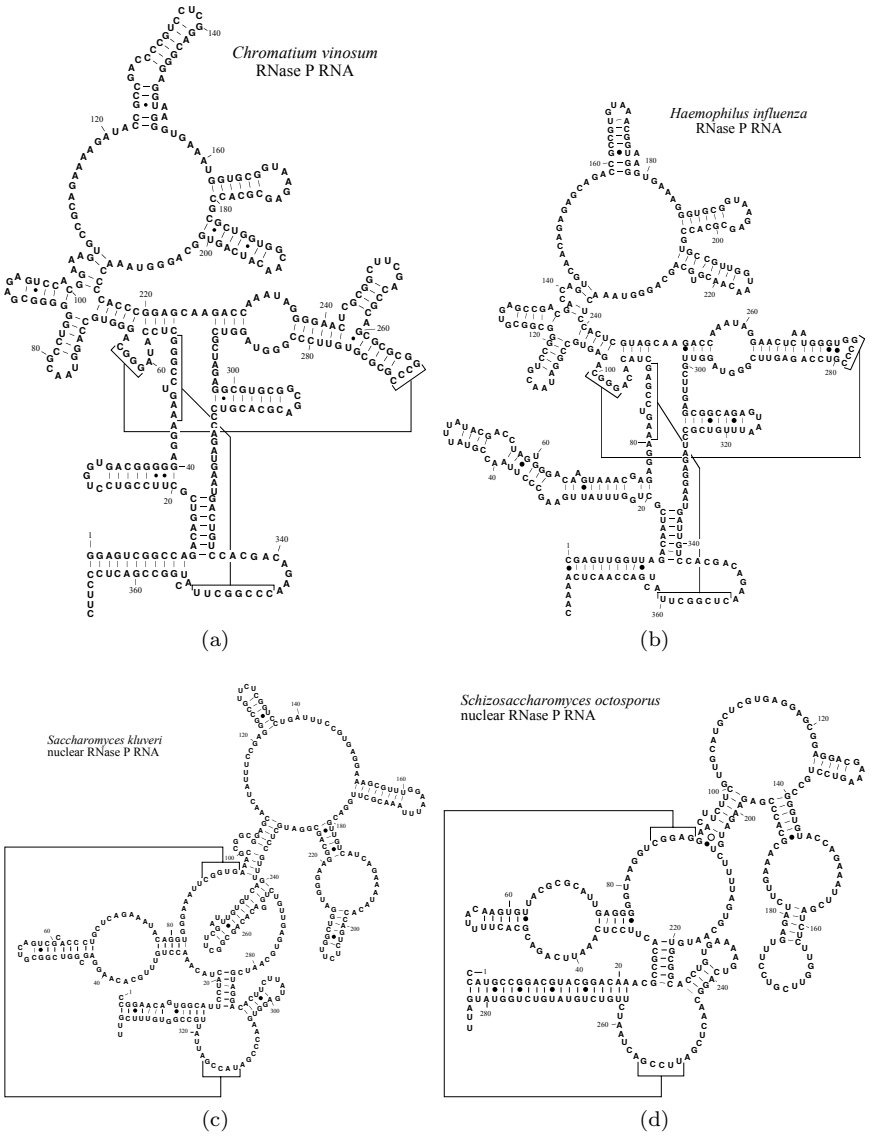


Figure 10: Secondary structure of the two RNA sequences involved in instances Real_1 (a and b) and Real_8 (c and d). All graphics were downloaded from the RNase P Database [31] (a) RNA of *Allochromatium vinosum*, (b) RNA of *Haemophilus influenza*, (c) RNA of *Saccharomyces kluveri*, (d) RNA of *Schizosaccharomyces octosporus*.